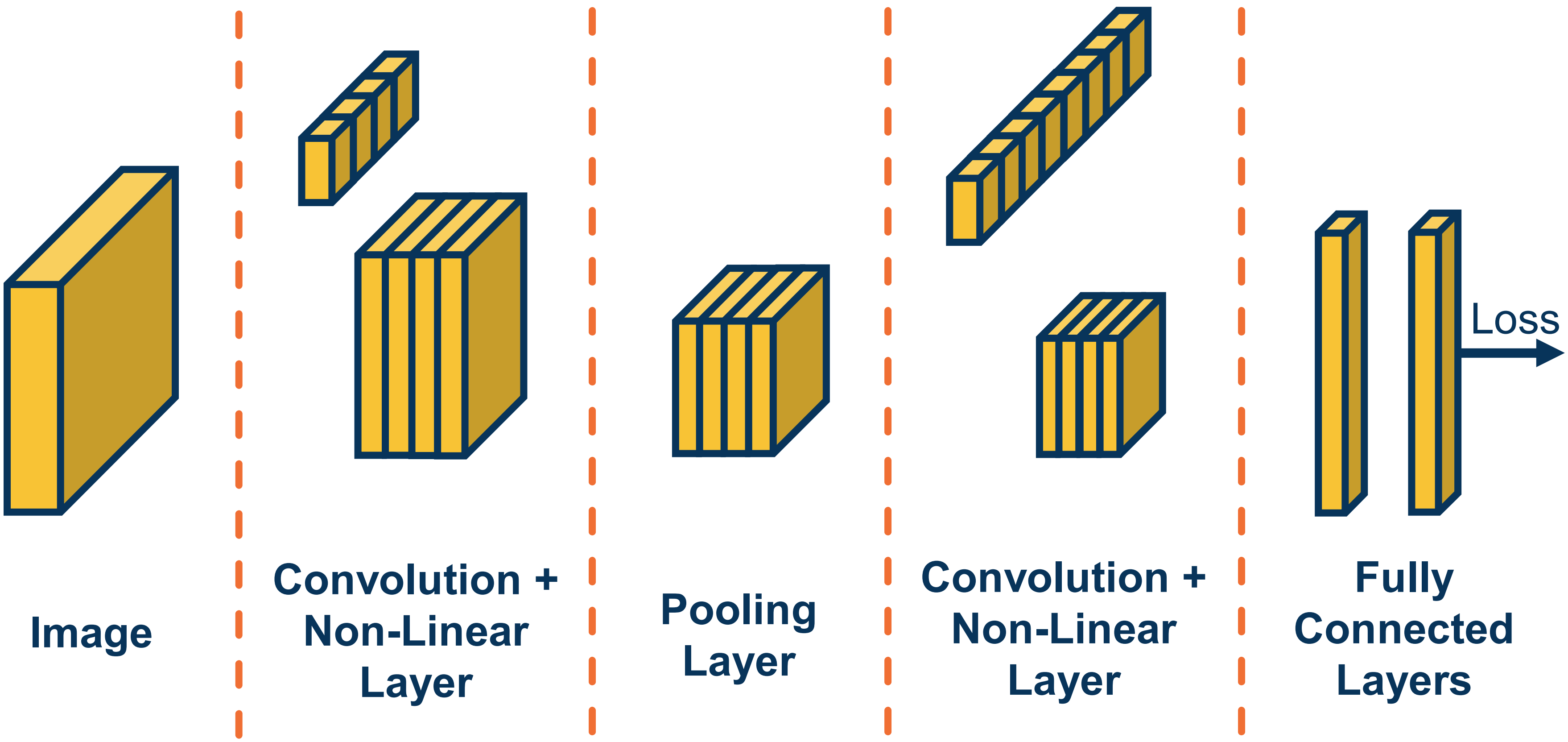
Topics:

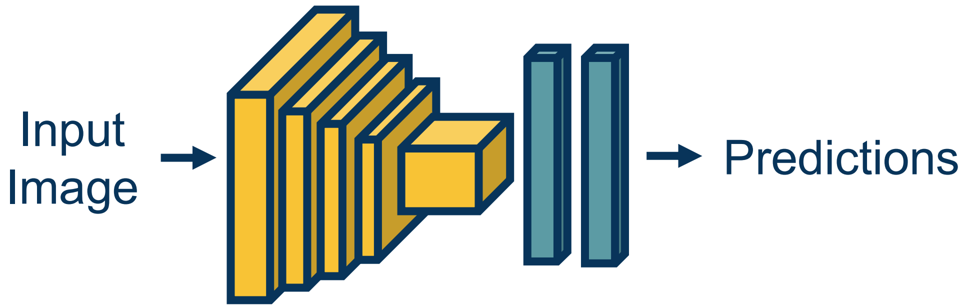
- Convolutional Neural Networks
- Visualization

CS 4644-DL / 7643-A
ZSOLT KIRA

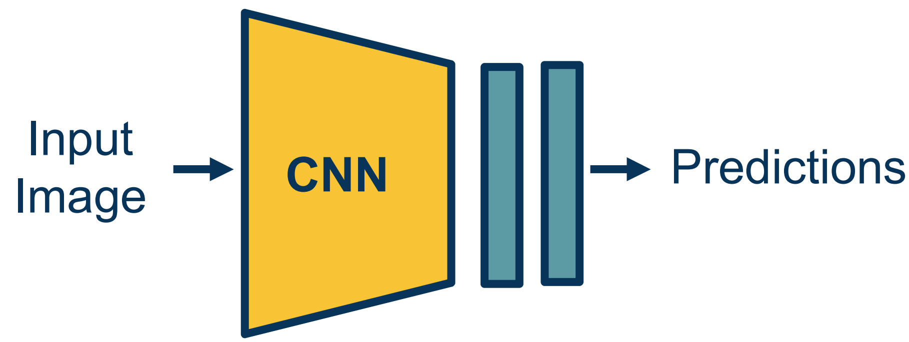
- **Assignment 2**
  - Due soon!
  - Resources (in addition to lectures):
    - DL book: Convolutional Networks
    - CNN notes https://www.cc.gatech.edu/classes/AY2022/cs7643_spring/assets/L10_cnns_notes.pdf
    - Backprop notes https://www.cc.gatech.edu/classes/AY2022/cs7643_spring/assets/L10_cnns_backprop_notes.pdf
    - **HW2 Tutorial @113, Conv @116, Focal Loss @117**
    - Slower OMSCS lectures on dropbox: Module 2 Lessons 5-6 (M2L5/M2L6) (https://www.dropbox.com/sh/iviro188gq0b4vs/AADdHxX_Uy1TkpF_yvIzX0nPa?dl=0)



Adding a Fully Connected Layer



Convolutional Neural Networks



Typical Depiction of CNNs

These architectures have existed **since 1980s**

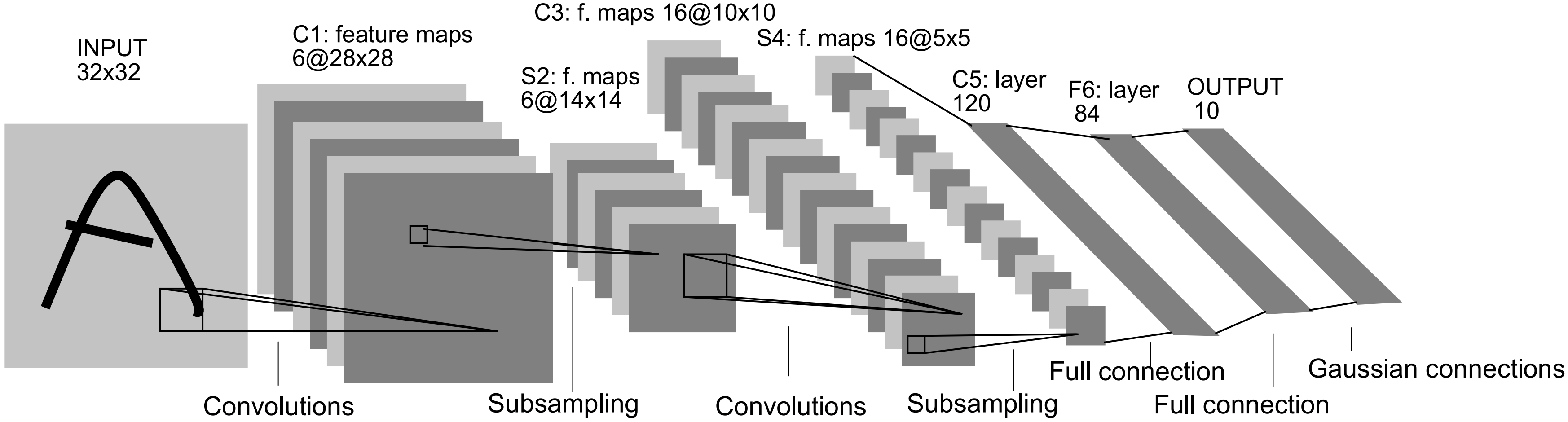


Image Credit: Yann LeCun, Kevin Murphy

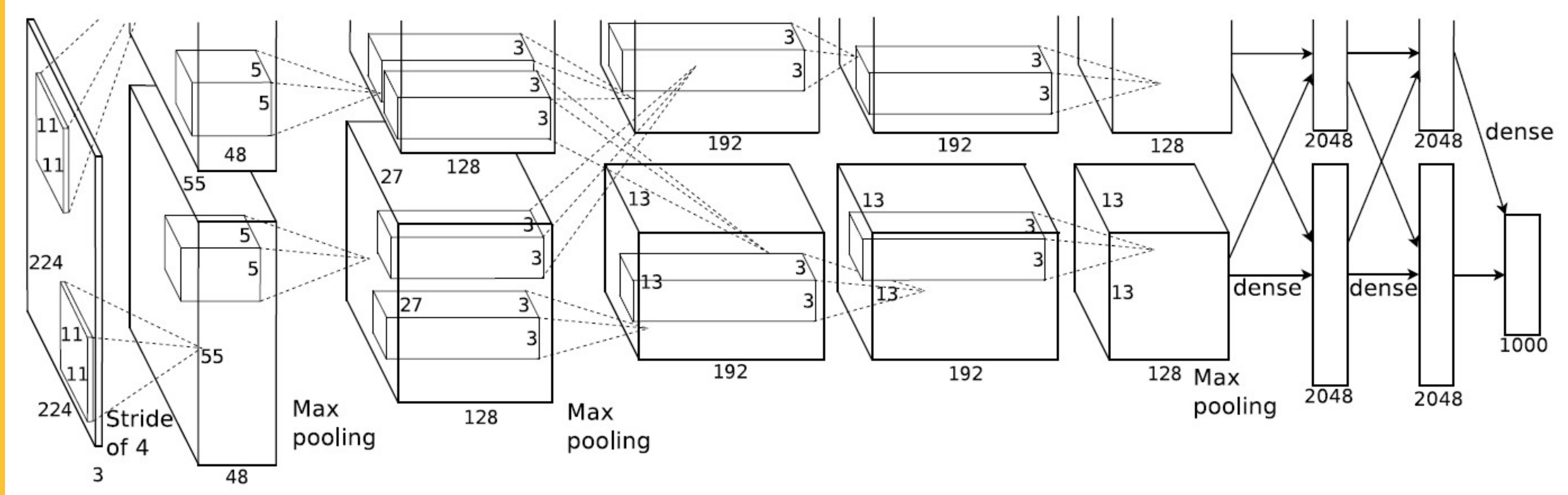
LeNet Architecture

The Importance of Benchmarks

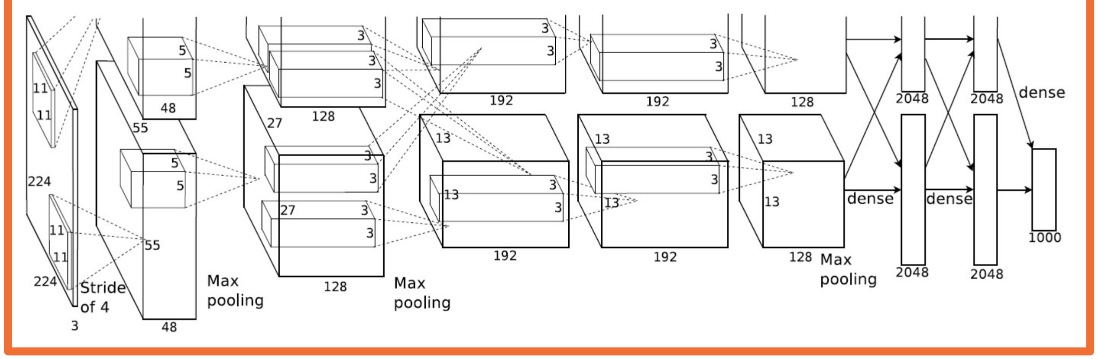
From: https://paperswithcode.com

Georgia Tech

AlexNet - Architecture



From: Krizhevsky et al., ImageNet Classification with Deep ConvolutionalNeural Networks, 2012.



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

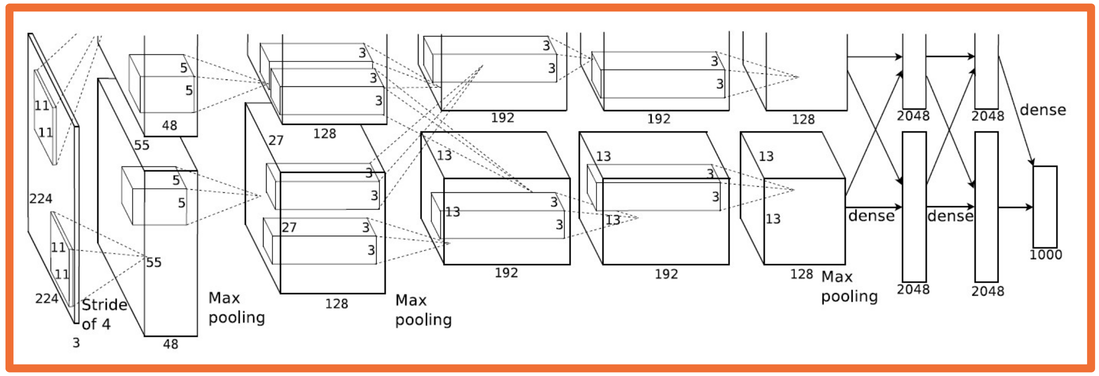
W' = (W - F + 2P) / S + 1

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

AlexNet – Layers and Key Aspects

Georgia Tech

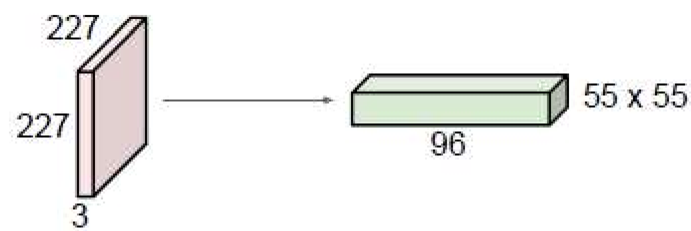
AlexNet – Layers and Key Aspects



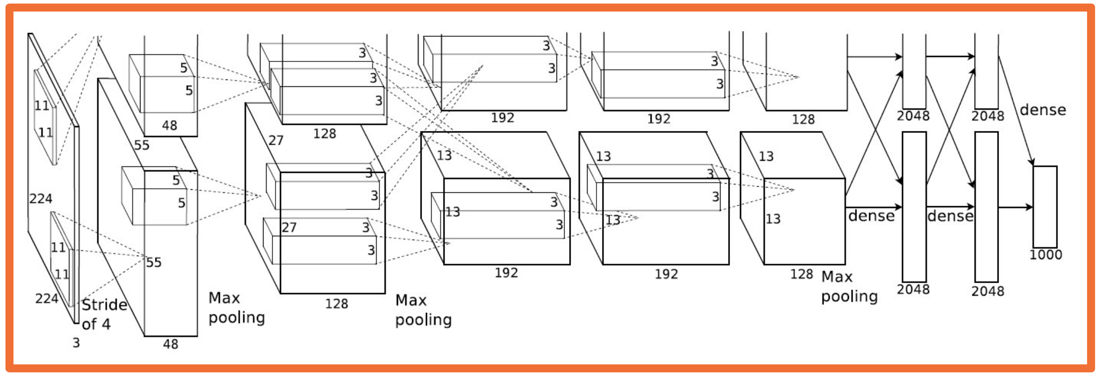
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

$$W' = (W - F + 2P) / S + 1$$



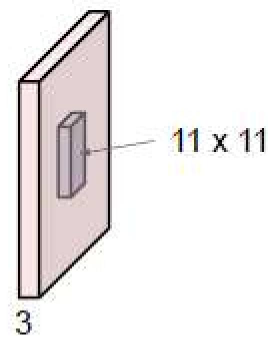
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



Input: 227x227x3 images

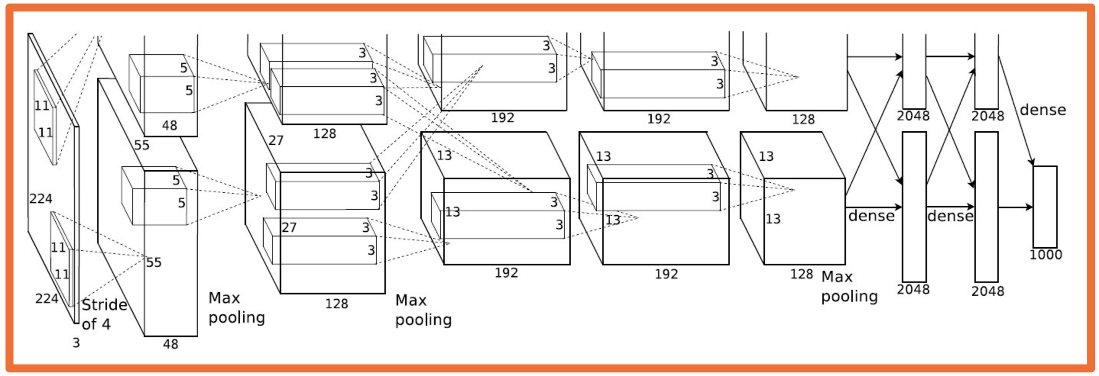
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?



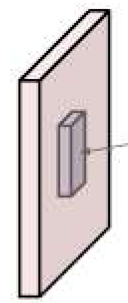
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

AlexNet – Layers and Key Aspects



Input: 227x227x3 images

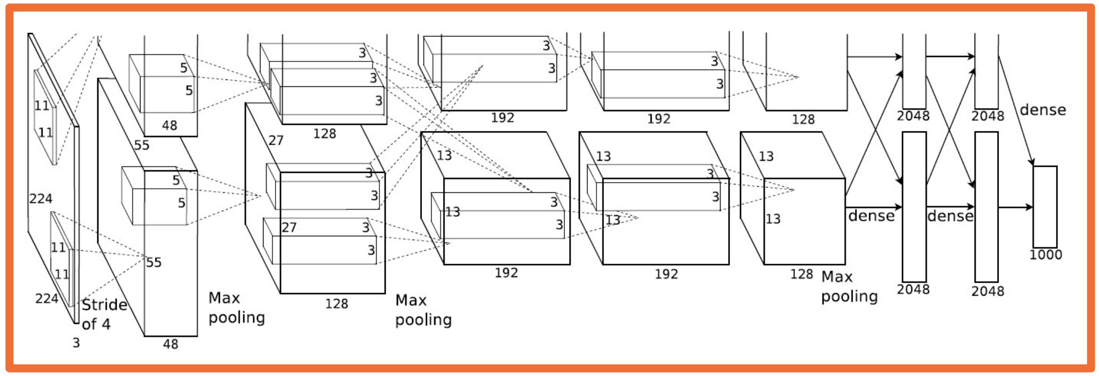
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3 + 1)*96 = **35K**



From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

AlexNet – Layers and Key Aspects

Full (simplified) AlexNet architecture:
[224x224x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)



Key aspects:

- ReLU instead of sigmoid or tanh
- Specialized normalization layers
- PCA-based data augmentation
- Dropout
- Ensembling

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

AlexNet – Layers and Key Aspects

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

VGGNet

Q: Why use smaller filters? (3x3 conv)

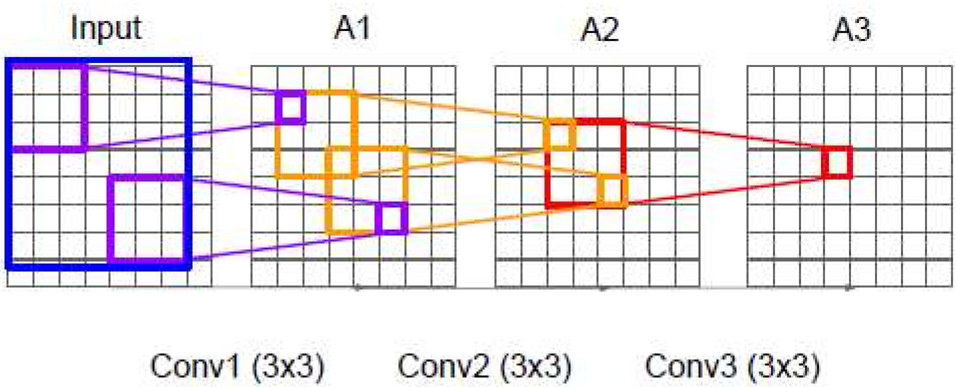
Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

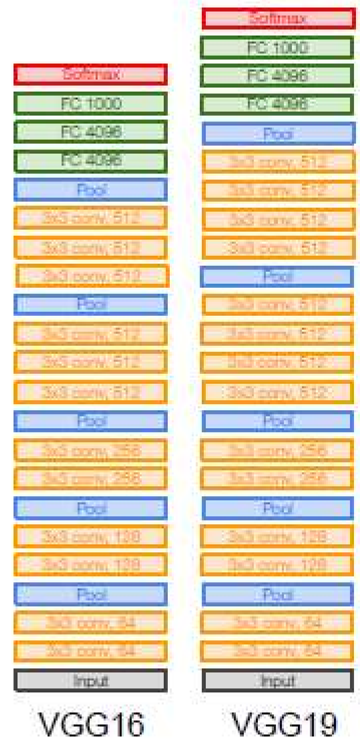
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

VGGNet

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?





From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

VGGNet

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

VGGNet

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

From: Simonyan & Zimmerman, Very Deep Convolutional Networks for Large-Scale Image Recognition
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

VGG

Parameters and Memory

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Most memory usage in convolution layers

Most parameters in FC layers

From: Simonyan & Zimmerman, Very Deep Convolutional Networks for Large-Scale Image Recognition
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Key aspects:

Repeated application of:

- 3x3 conv (stride of 1, padding of 1)
- 2x2 max pooling (stride 2)

Very large number of parameters (138M)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

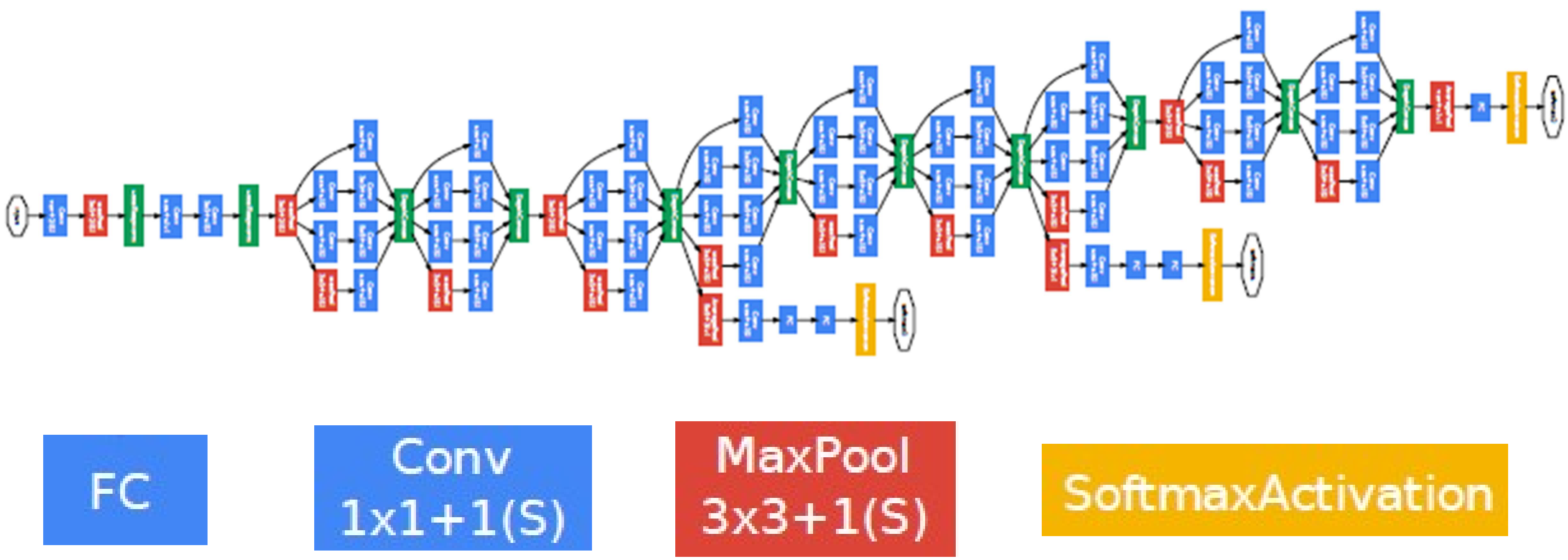
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

From: Simonyan & Zimmerman, Very Deep Convolutional Networks for Large-Scale Image Recognition
From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

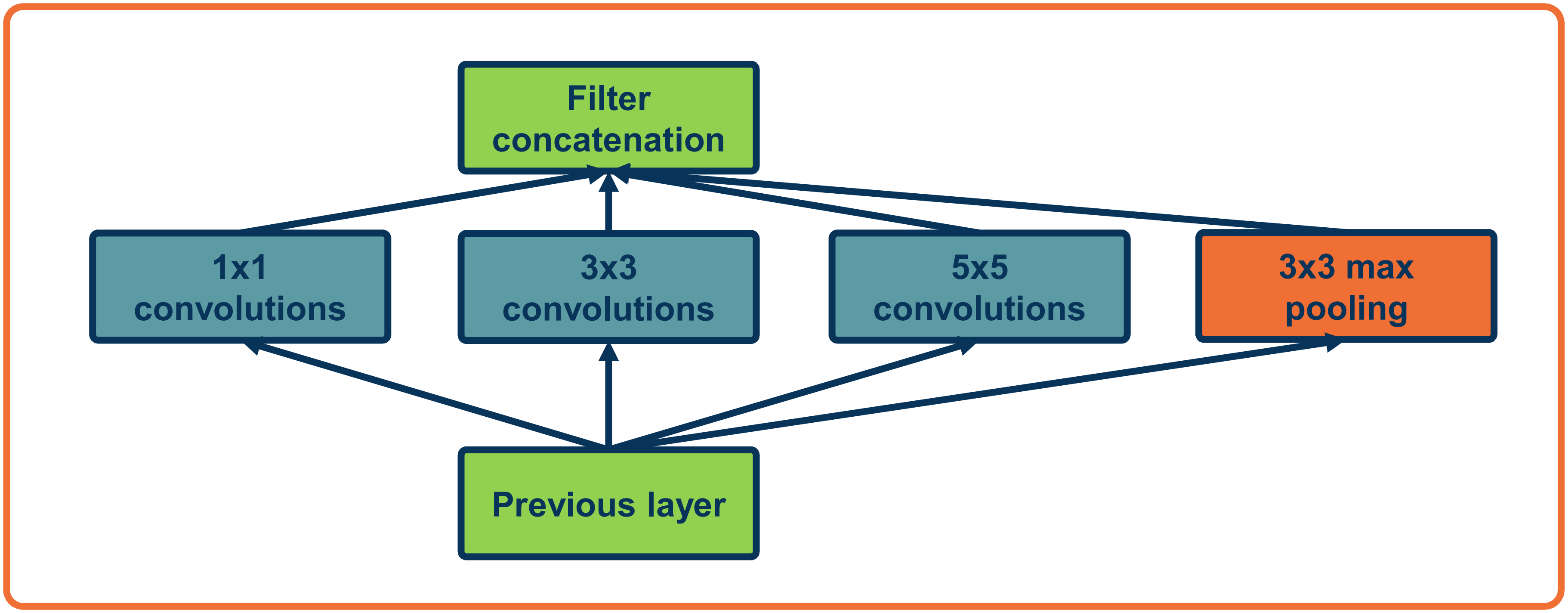
VGG – Key Characteristics

But have become **deeper and more complex**



From: Szegedy et al. Going deeper with convolutions

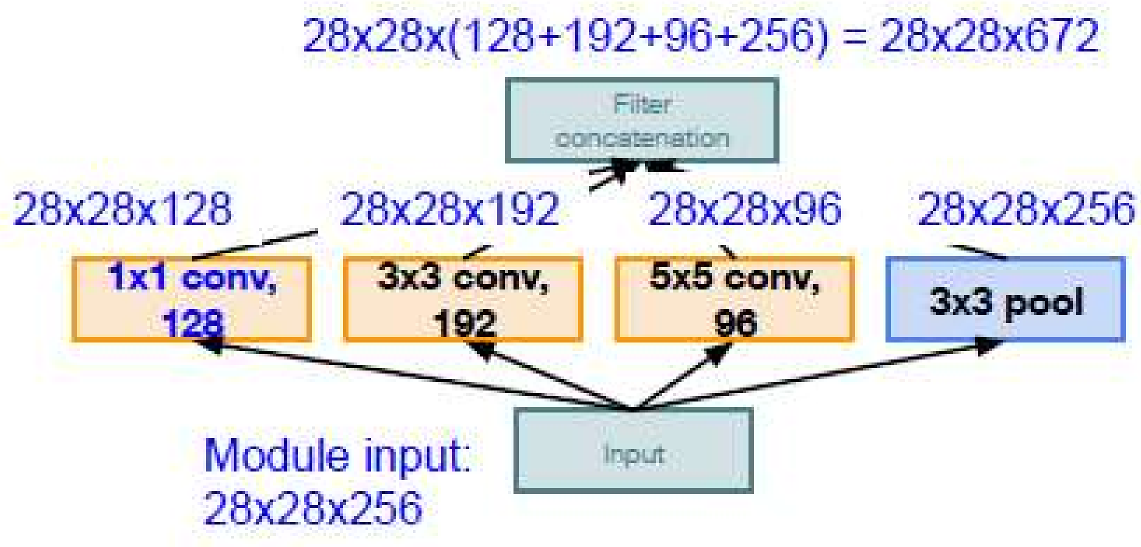
Key idea: Repeated blocks and multi-scale features



From: Szegedy et al. Going deeper with convolutions

Inception Module

Key idea: Repeated blocks and multi-scale features

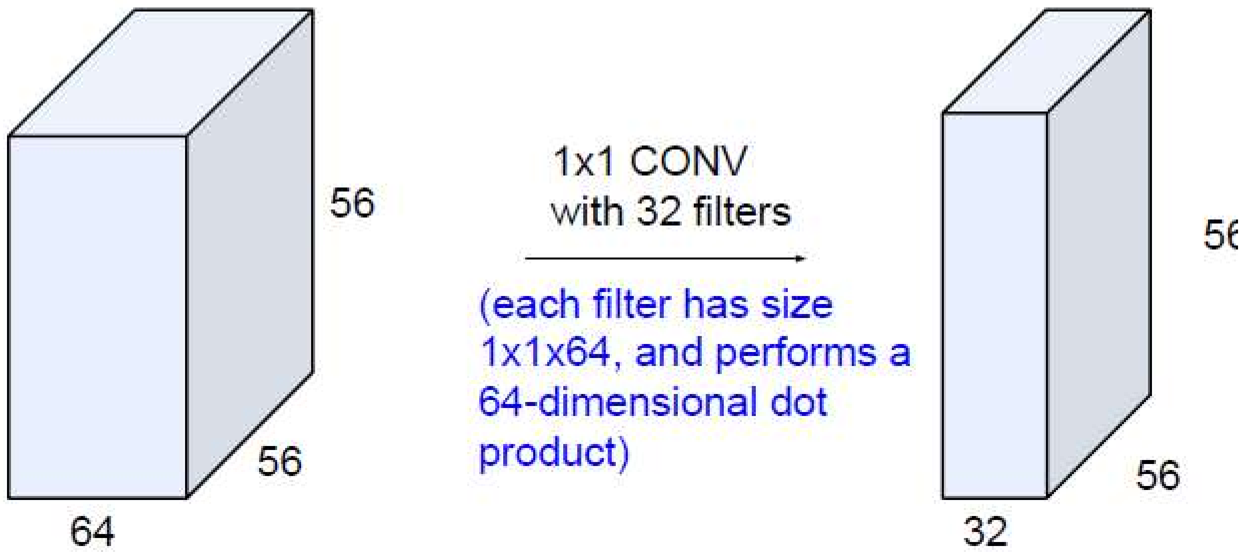


Naive Inception module

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

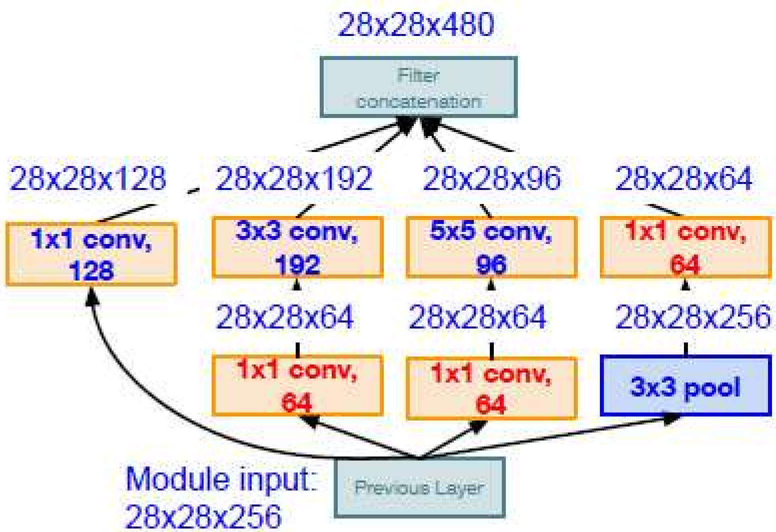
Inception Module

Apply 1x1 convolutions as bottleneck layer (decrease number of channels!)



From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:



Inception module with dimension reduction

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

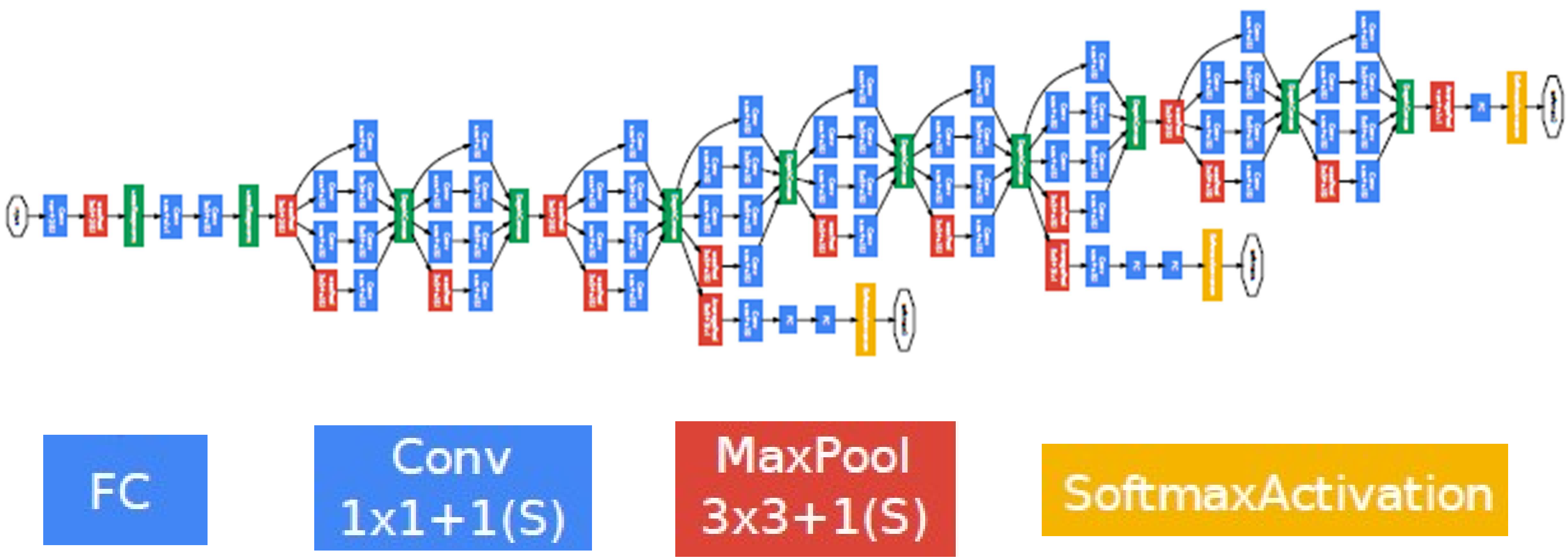
Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

From: Slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Inception Module

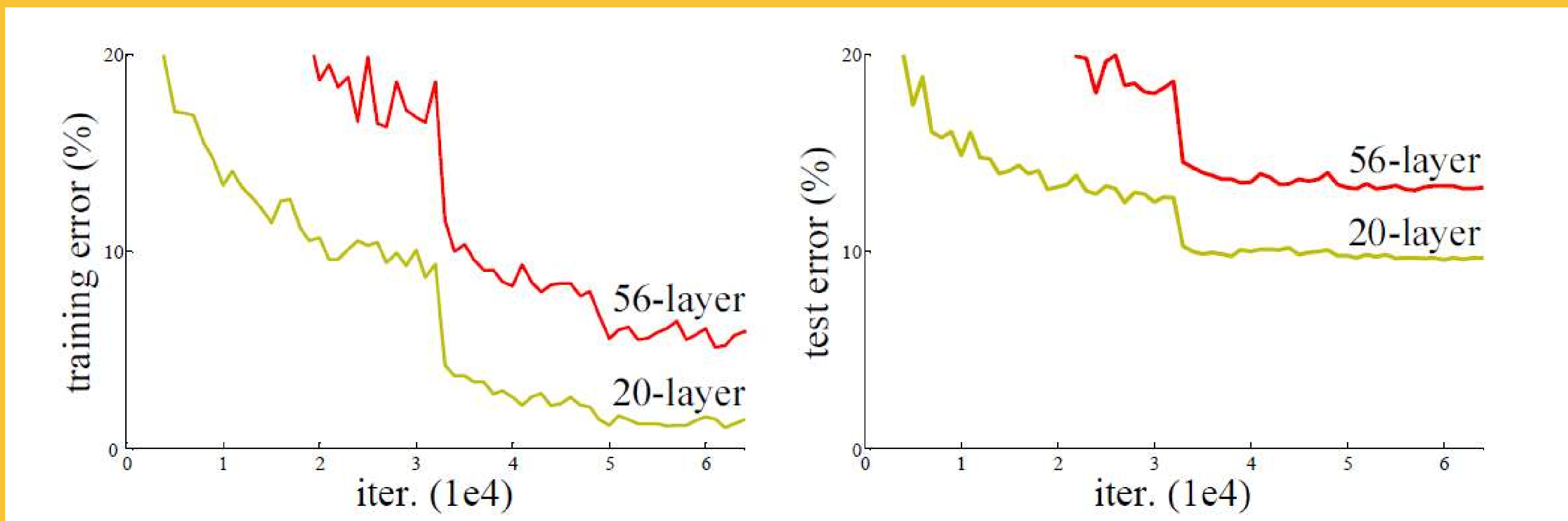
Georgia Tech

But have become **deeper and more complex**



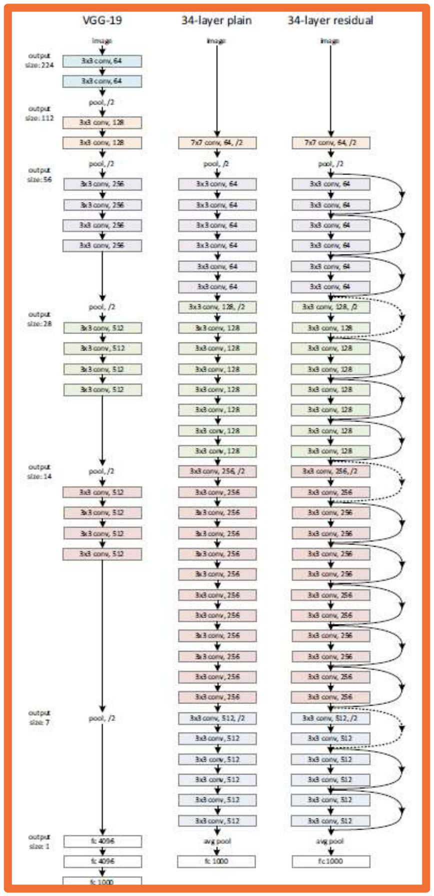
From: Szegedy et al. Going deeper with convolutions

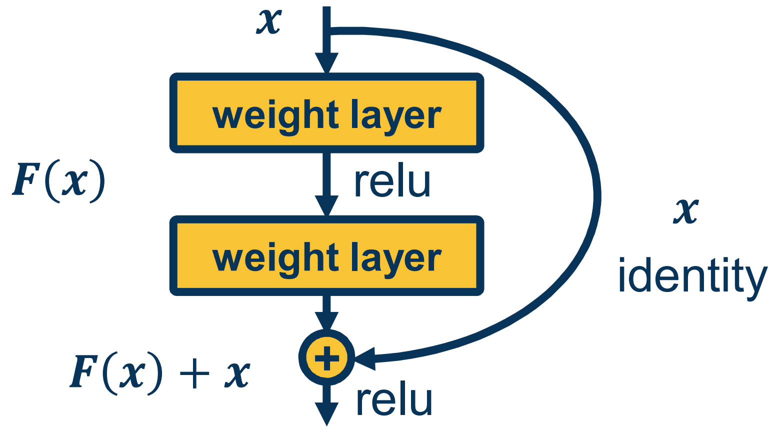
The Challenge of Depth



From: He et al., Deep Residual Learning for Image Recognition

Optimizing very deep networks is challenging!





Key idea: Allow information from a layer to propagate to any future layer (forward)

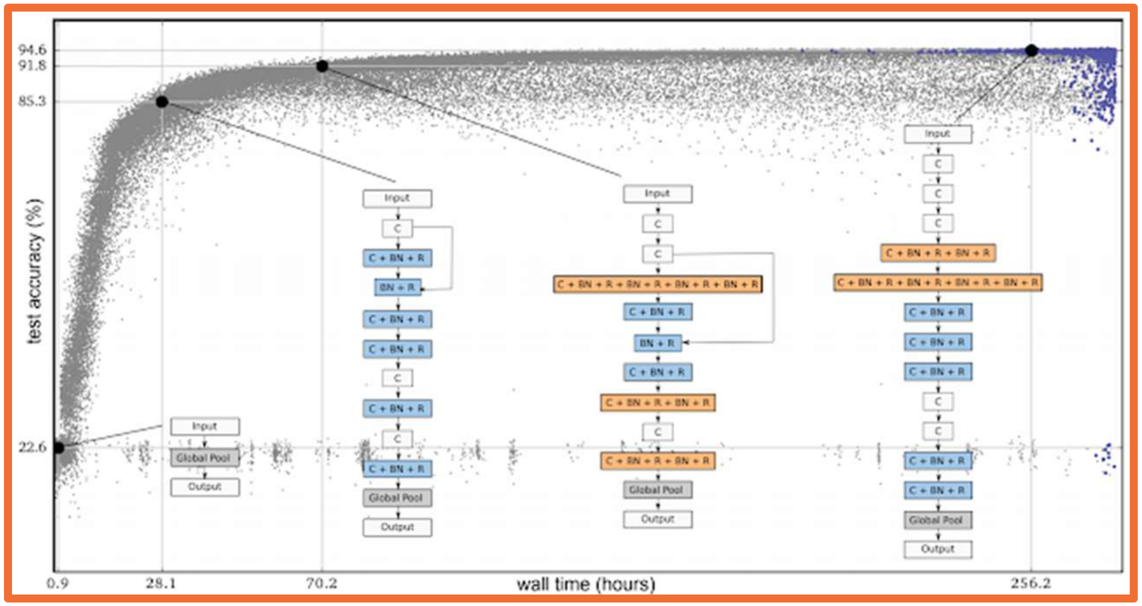
Same is true for gradients!

From: He et al., Deep Residual Learning for Image Recognition

Residual Blocks and Skip Connections

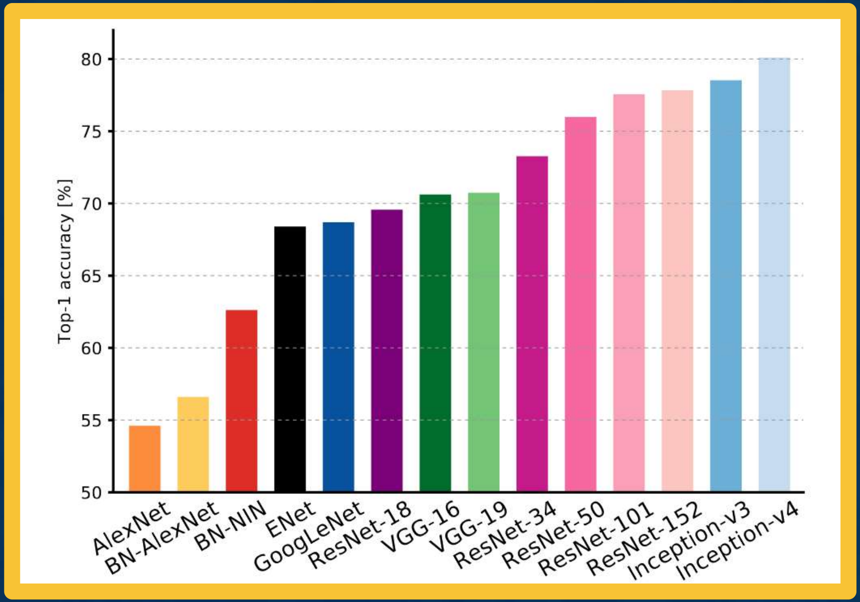
Several ways to *learn* architectures:

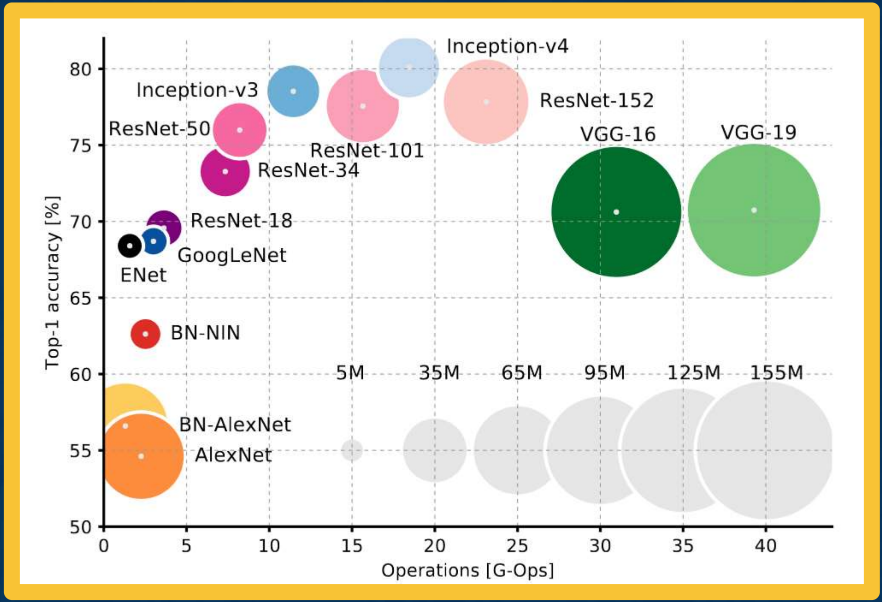
- Evolutionary learning and reinforcement learning
- Prune over-parameterized networks
- Learning of **repeated blocks** typical



From: https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html

Computational Complexity

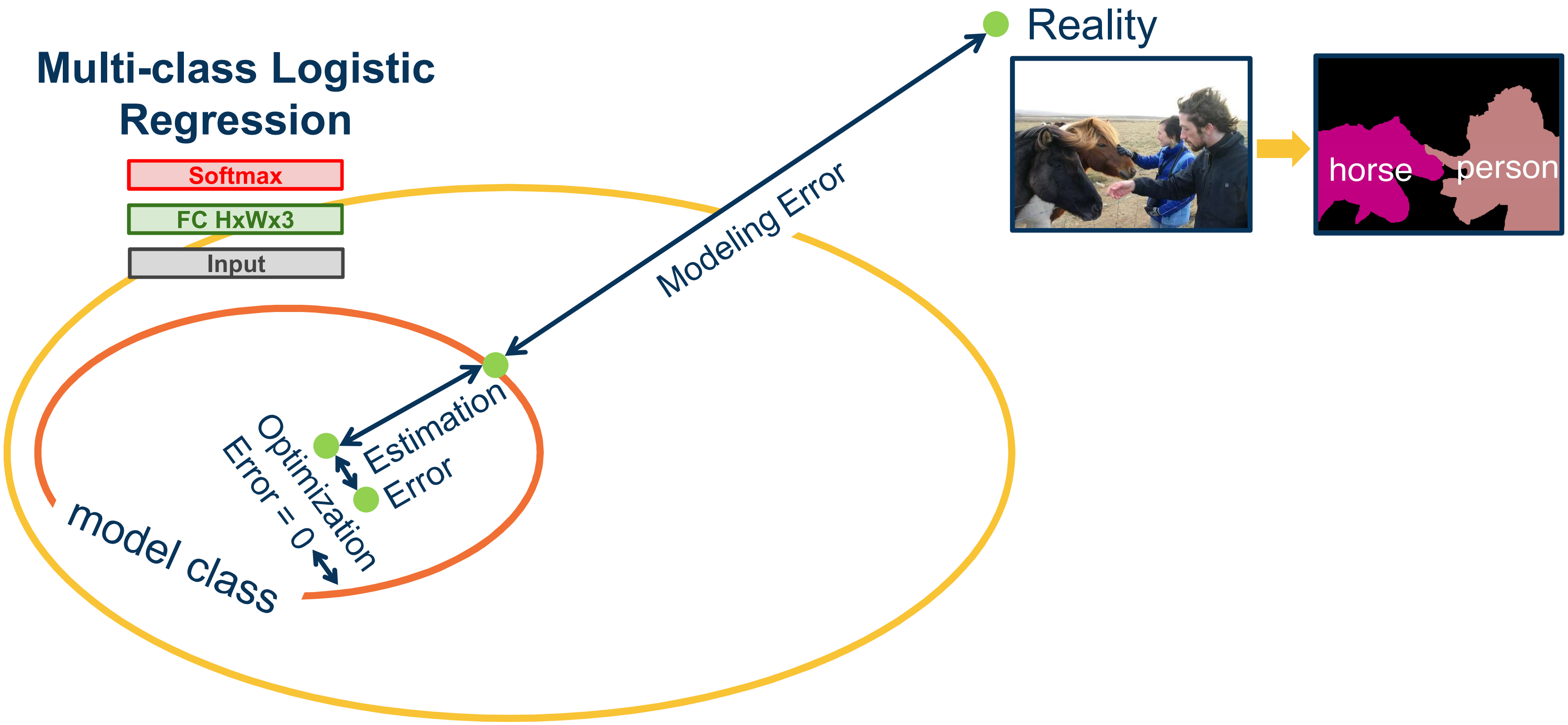




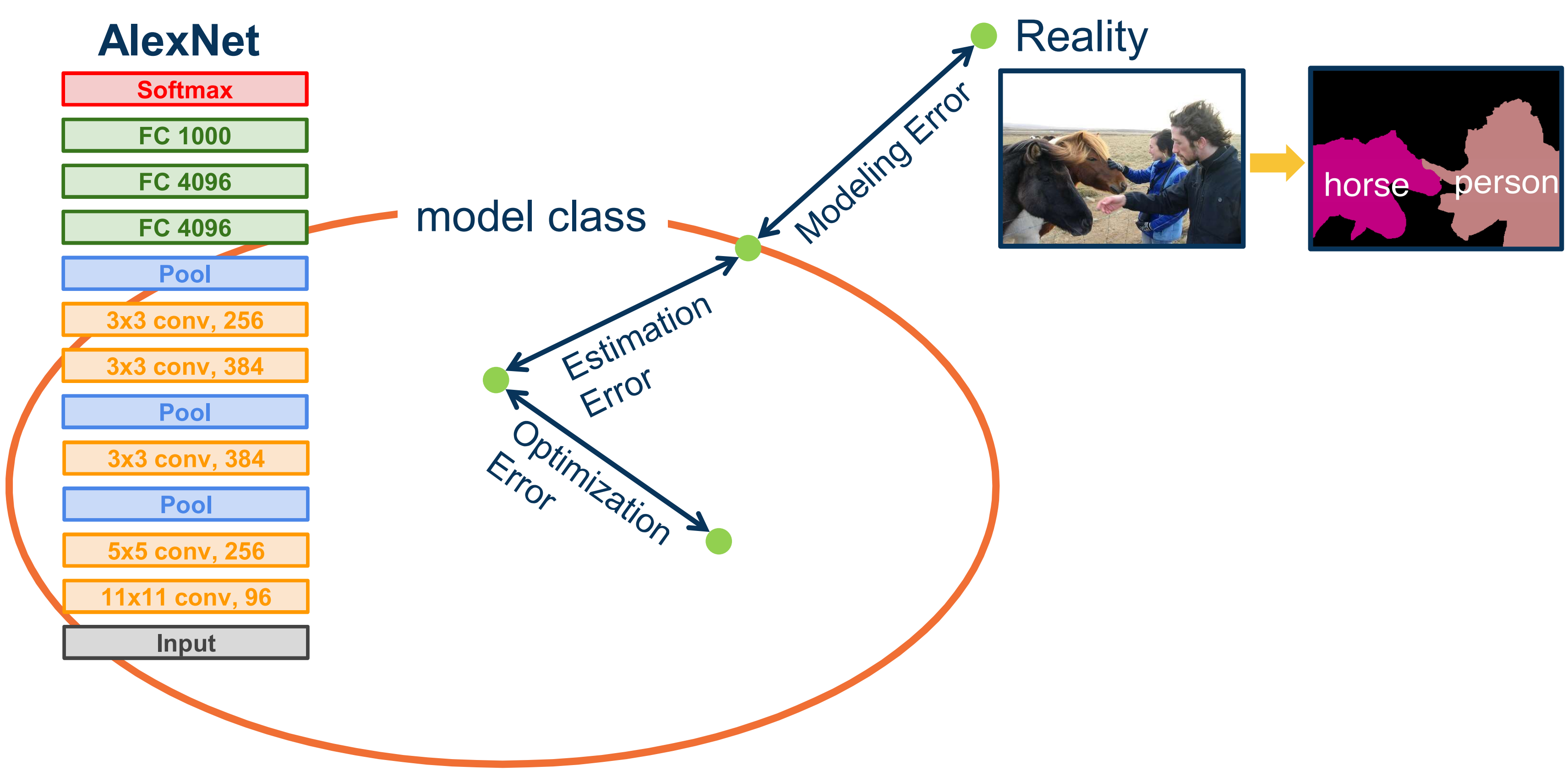
From: An Analysis Of Deep Neural Network Models For Practical Applications

Georgia Tech

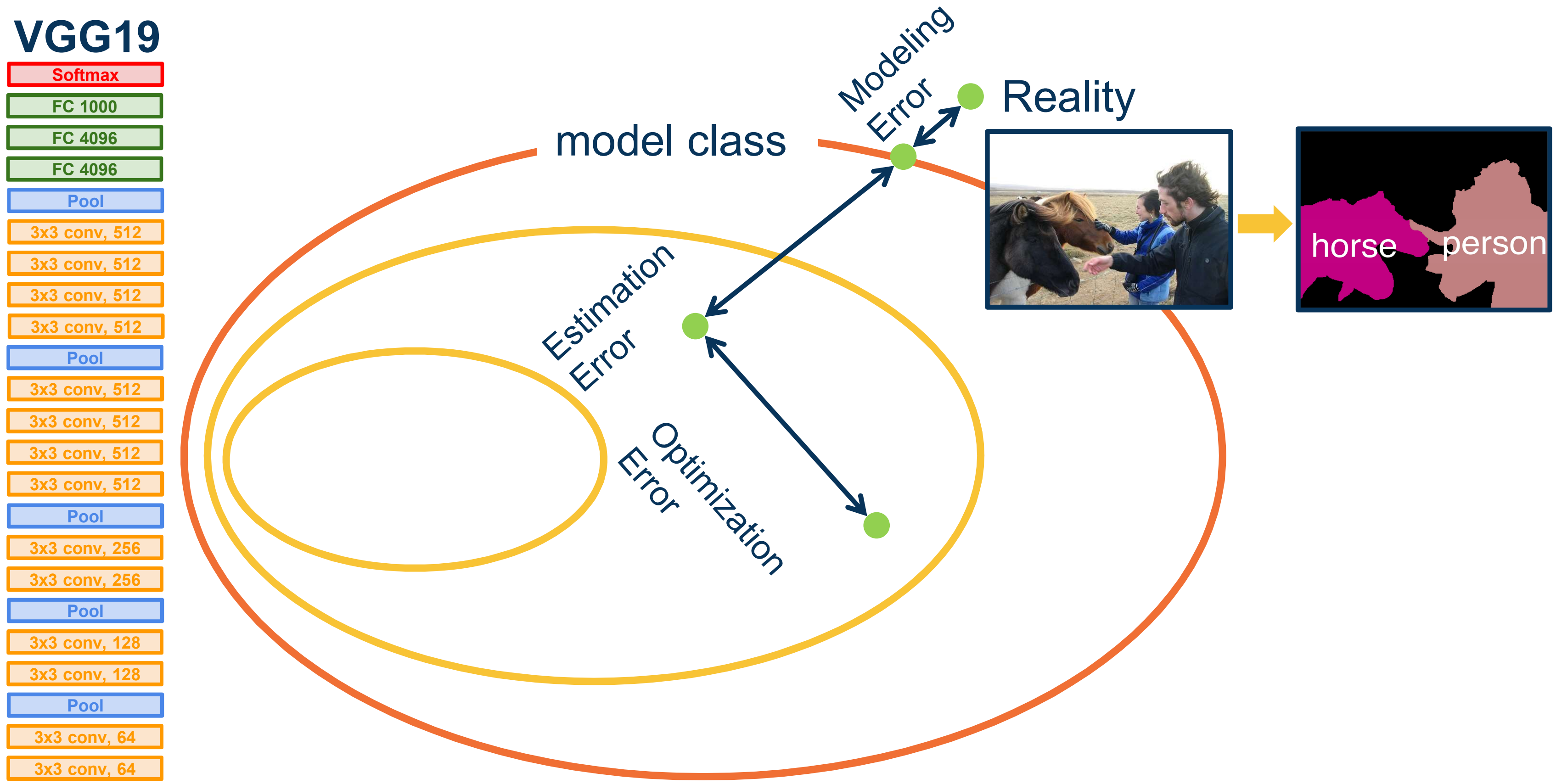
Transfer Learning & Generalization



From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



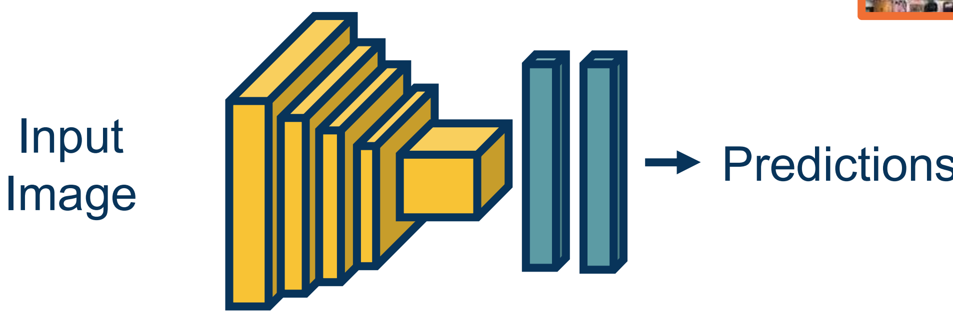
From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



From: slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

What if we don't have enough data?

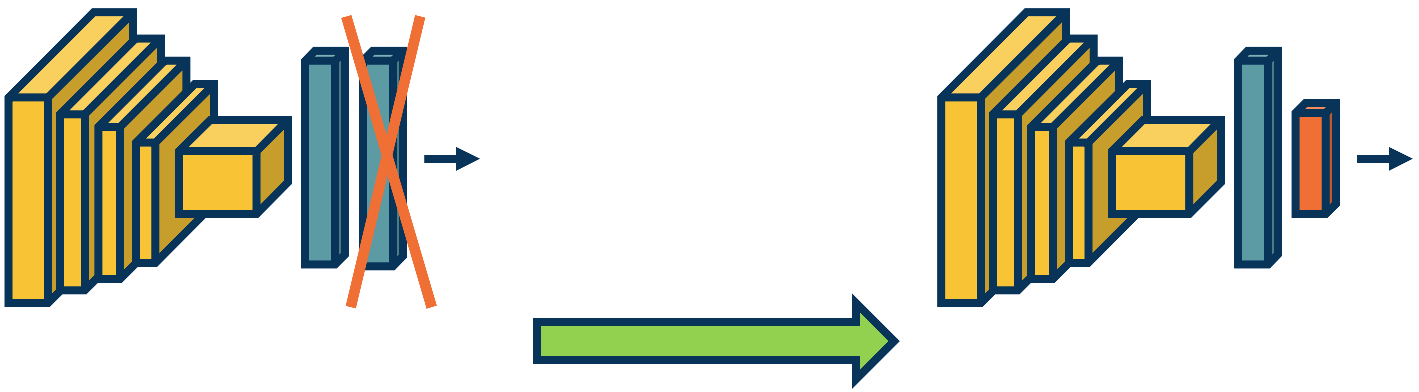
Step 1: Train on large-scale dataset



Convolutional Neural Networks

Transfer Learning – Training on Large Dataset

Step 2: Take your custom data and **initialize** the network with weights trained in Step 1



Replace last layer with new fully-connected for output nodes per new category

Step 3: (Continue to) train on new dataset

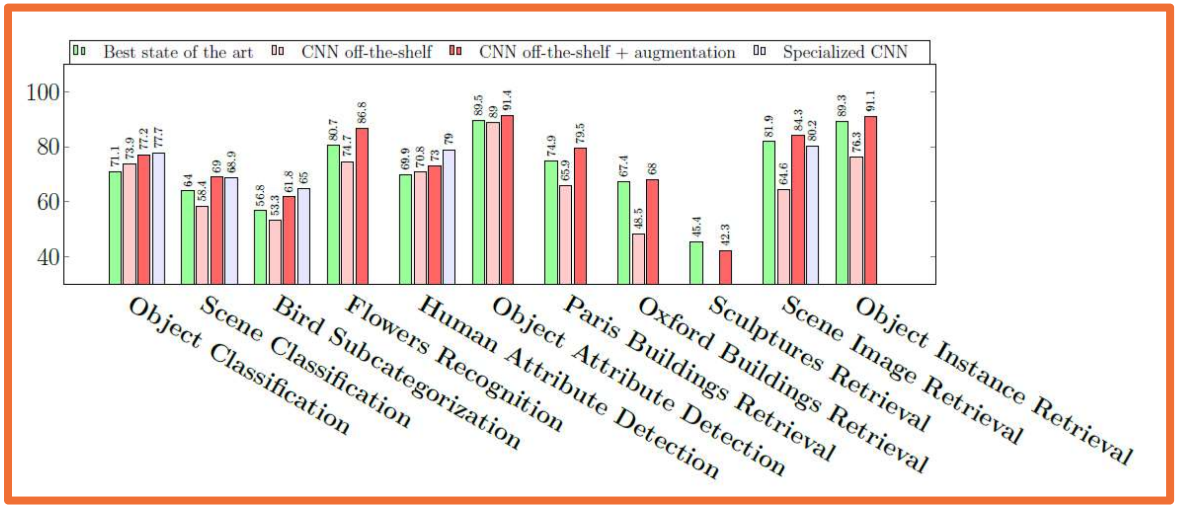
- **Finetune:** Update all parameters
- **Freeze** feature layer: Update only last layer weights (used when not enough data)

Replace last layer with new fully-connected for output nodes per new category

Finetuning on New Dataset

This works extremely well! It was surprising upon discovery.

- Features learned for 1000 object categories will work well for 1001^{st}!
- Generalizes even across tasks (classification to object detection)



From: Razavian et al., CNN Features off-the-shelf: an Astounding Baseline for Recognition

Surprising Effectiveness of Transfer Learning

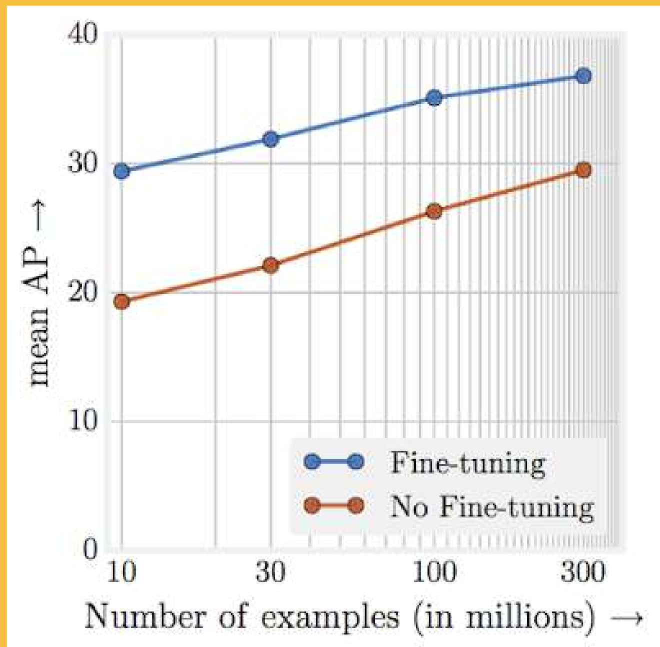
Georgia Tech

Learning with Less Labels

But it doesn't always work that well!

- If the **source** dataset you train on is very different from the **target** dataset, transfer learning is not as effective
- If you have enough data for the target domain, it just results in faster convergence
  - See He et al., "Rethinking ImageNet Pre-training"

Effectiveness of More Data



From: Revisiting the Unreasonable Effectiveness of Data
https://ai.googleblog.com/2017/07/revisiting-unreasonable-effectiveness.html

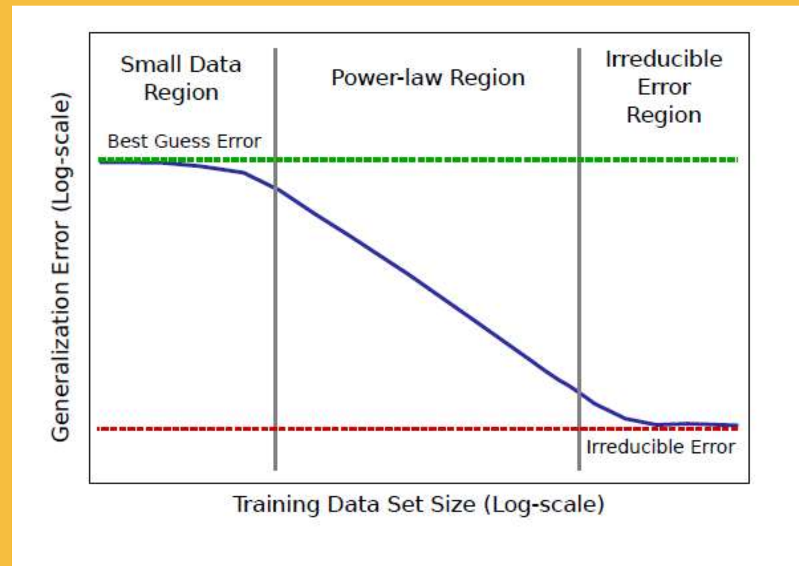


Figure 6: Sketch of power-law learning curves

From: Hestness et al., Deep Learning Scaling Is Predictable

There is a large number of different low-labeled settings in DL research

Setting	Source	Target	Shift Type
Semi-supervised	Single labeled	Single unlabeled	None
Domain Adaptation	Single labeled	Single unlabeled	Non-semantic
Domain Generalization	Multiple labeled	Unknown	Non-semantic
Cross-Task Transfer	Single labeled	Single unlabeled	Semantic
Few-Shot Learning	Single labeled	Single few-labeled	Semantic
Un/Self-Supervised	Single unlabeled	Many labeled	Both/Task

Non-Semantic Shift

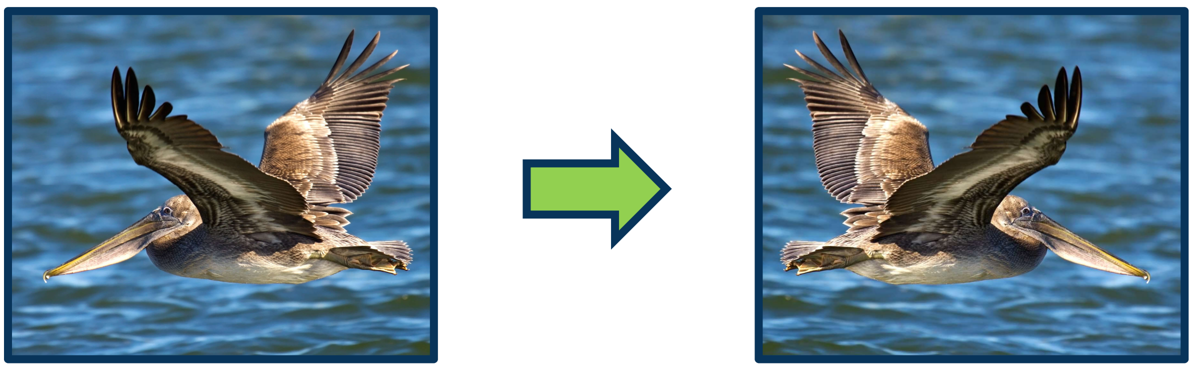
Semantic Shift

Data Augmentation

Data augmentation – Performing a range of **transformations** to the data

- This essentially **“increases”** your dataset
- Transformations should not change meaning of the data (or label has to be changed as well)

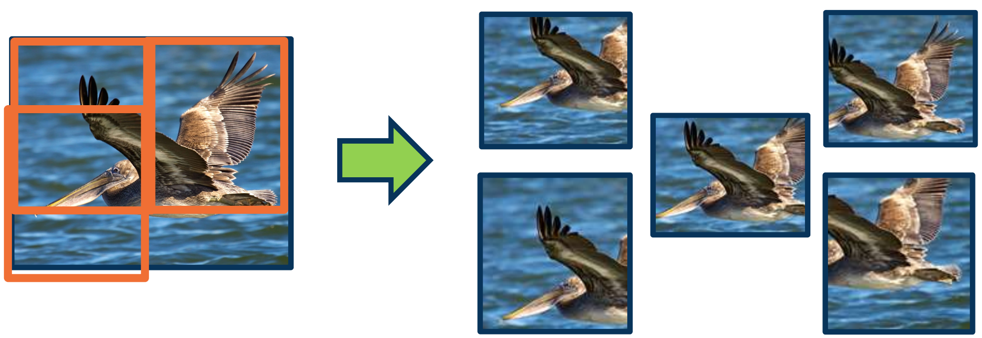
Simple example: Image Flipping



Data Augmentation: Motivation

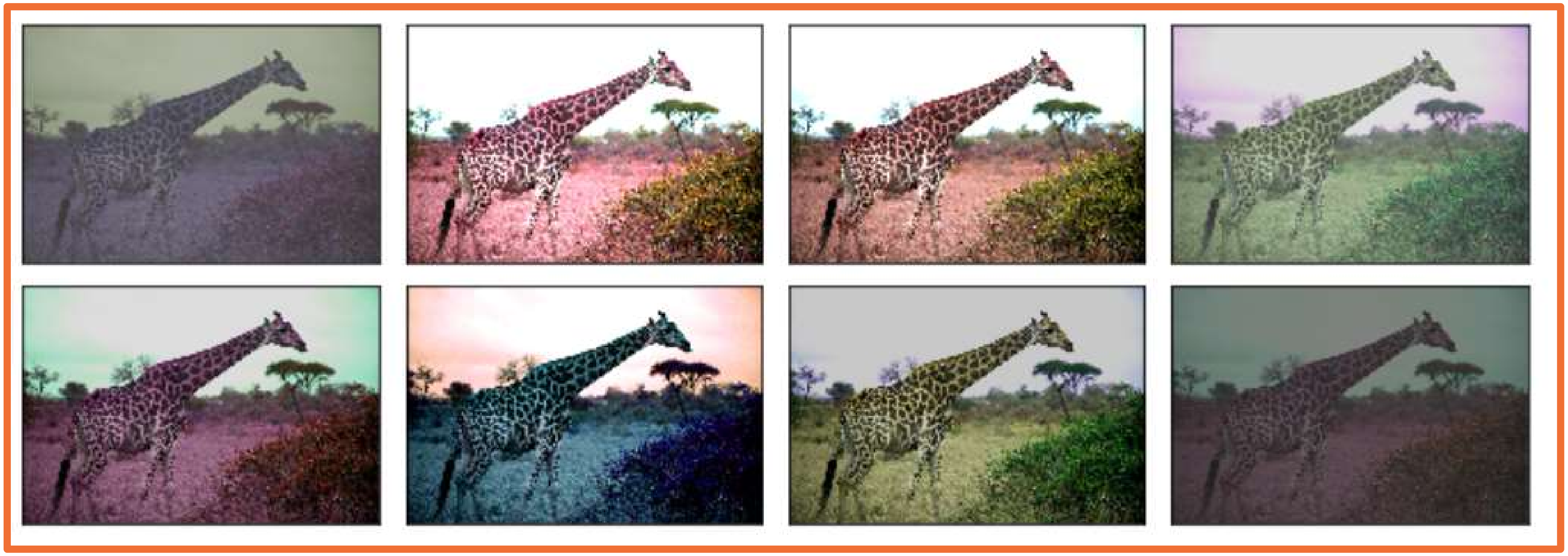
Random crop

- Take different crops during training
- Can be used during inference too!



CutMix

Color Jitter

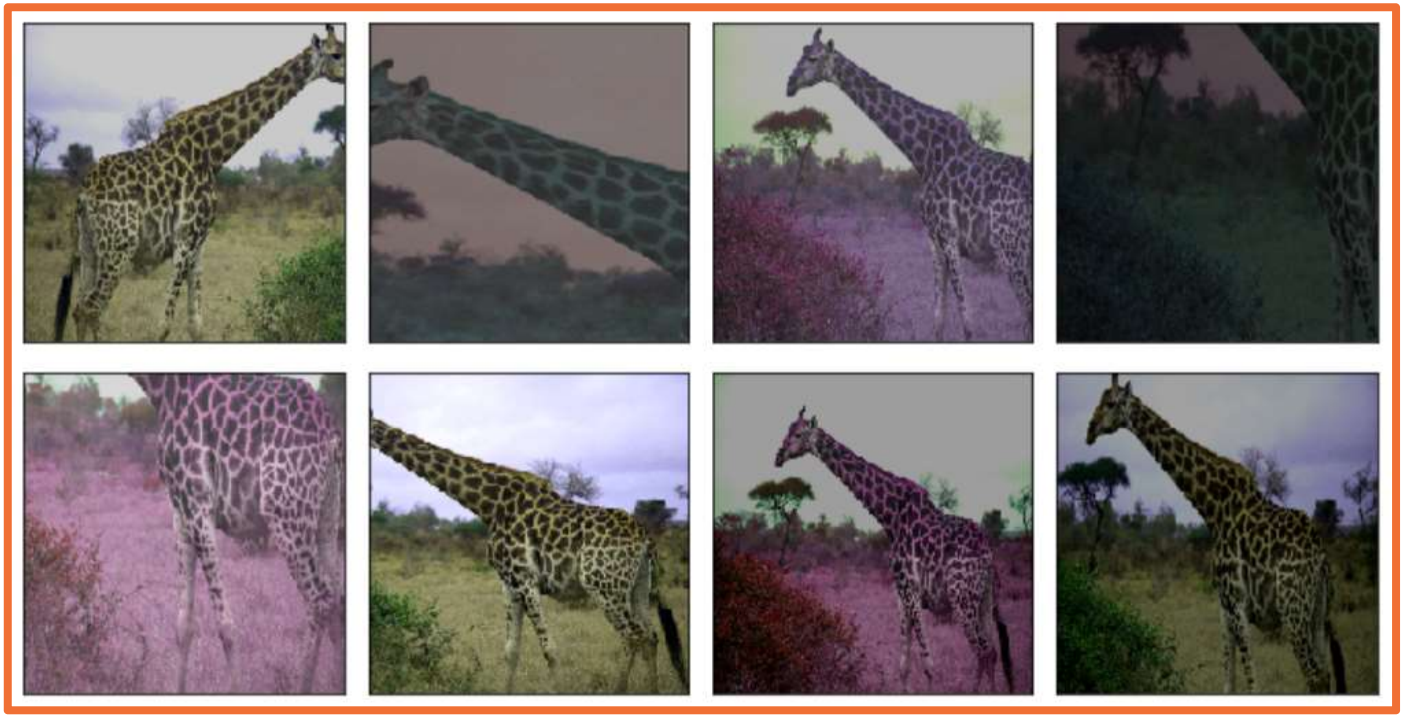


From https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html

We can apply **generic affine transformations:**

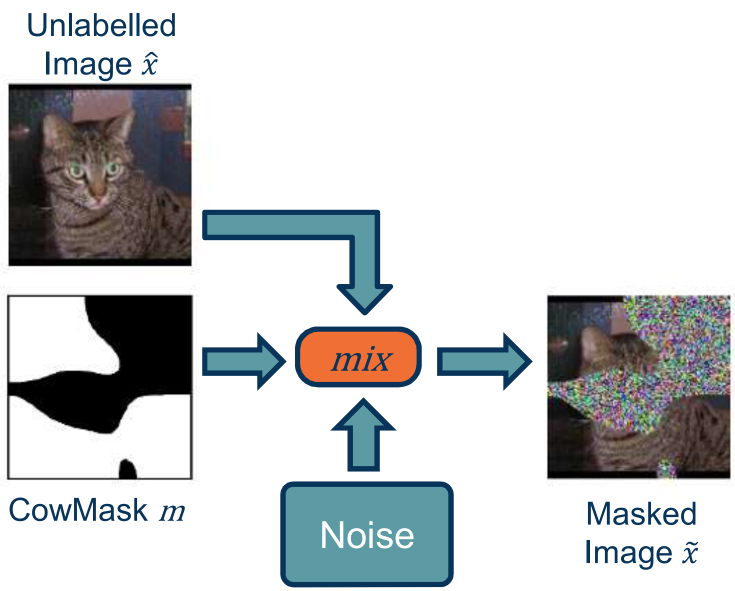
- **Translation**
- **Rotation**
- **Scale**
- **Shear**

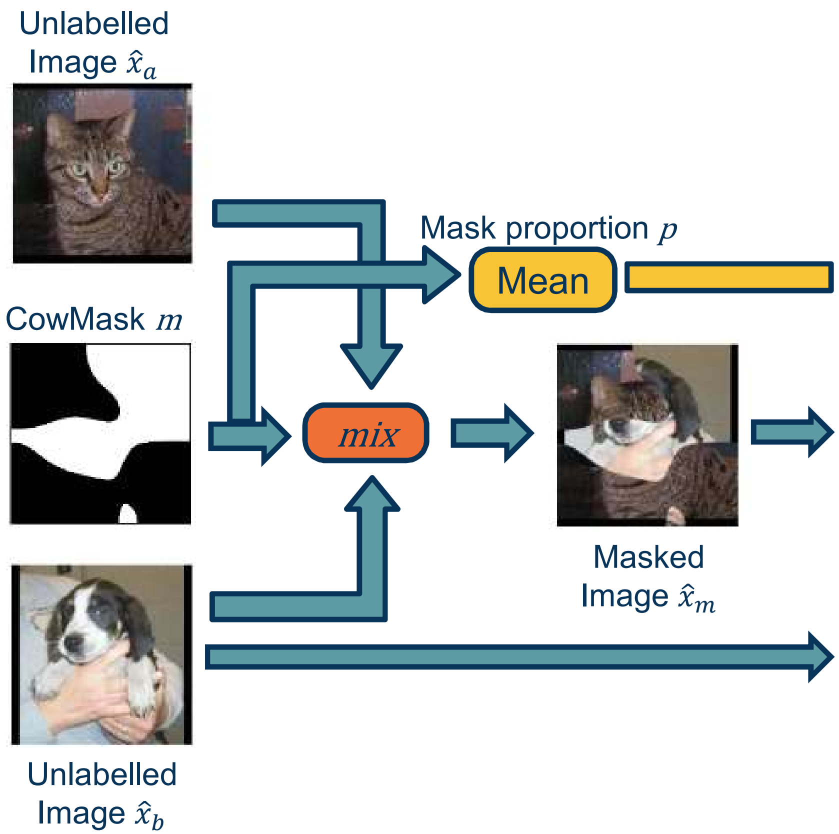
We can **combine these transformations** to add even more variety!



From https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html

Combining Transformations





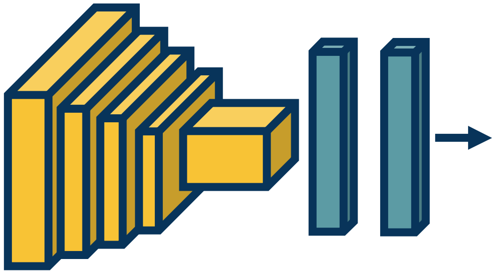
CowMix

From French et al., "Milking CowMask for Semi-Supervised Image Classification"

Visualization of Neural Networks

Georgia Tech

Given a **trained** model, we'd like to understand what it learned.



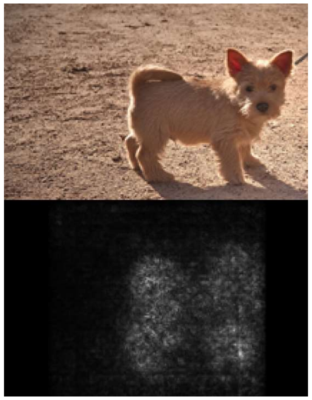
Weights

Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Zeiler & Fergus, 2014

Activations

Gradients



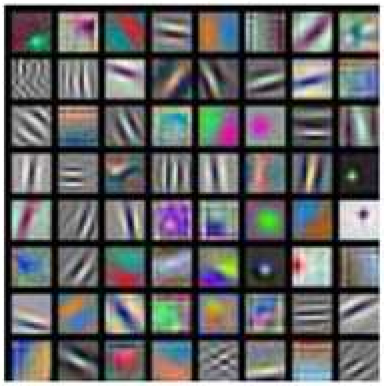
Simonyan et al, 2013

Robustness

Hendrycks & Dietterich, 2019

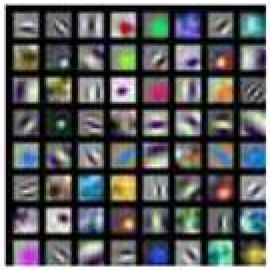
FC Layer: Reshape weights for a node back into size of image, scale 0-255

Conv layers:
For each kernel, scale values from 0-255 and visualize



AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7



ResNet-101:
64 x 3 x 7 x 7

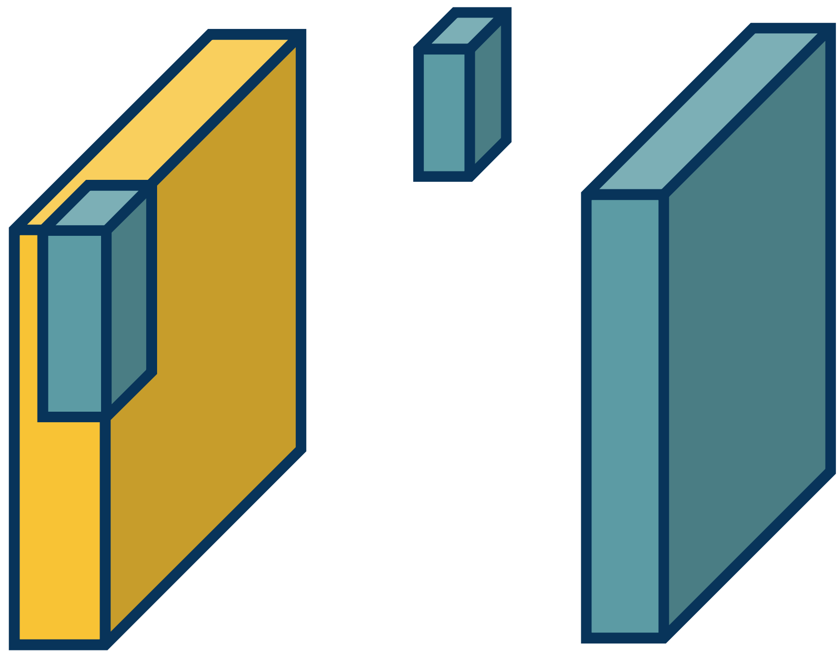
Problem: 3x3 filters difficult to interpret!

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Visualizing Weights

We can also produce **visualization output (aka activation/filter) maps**

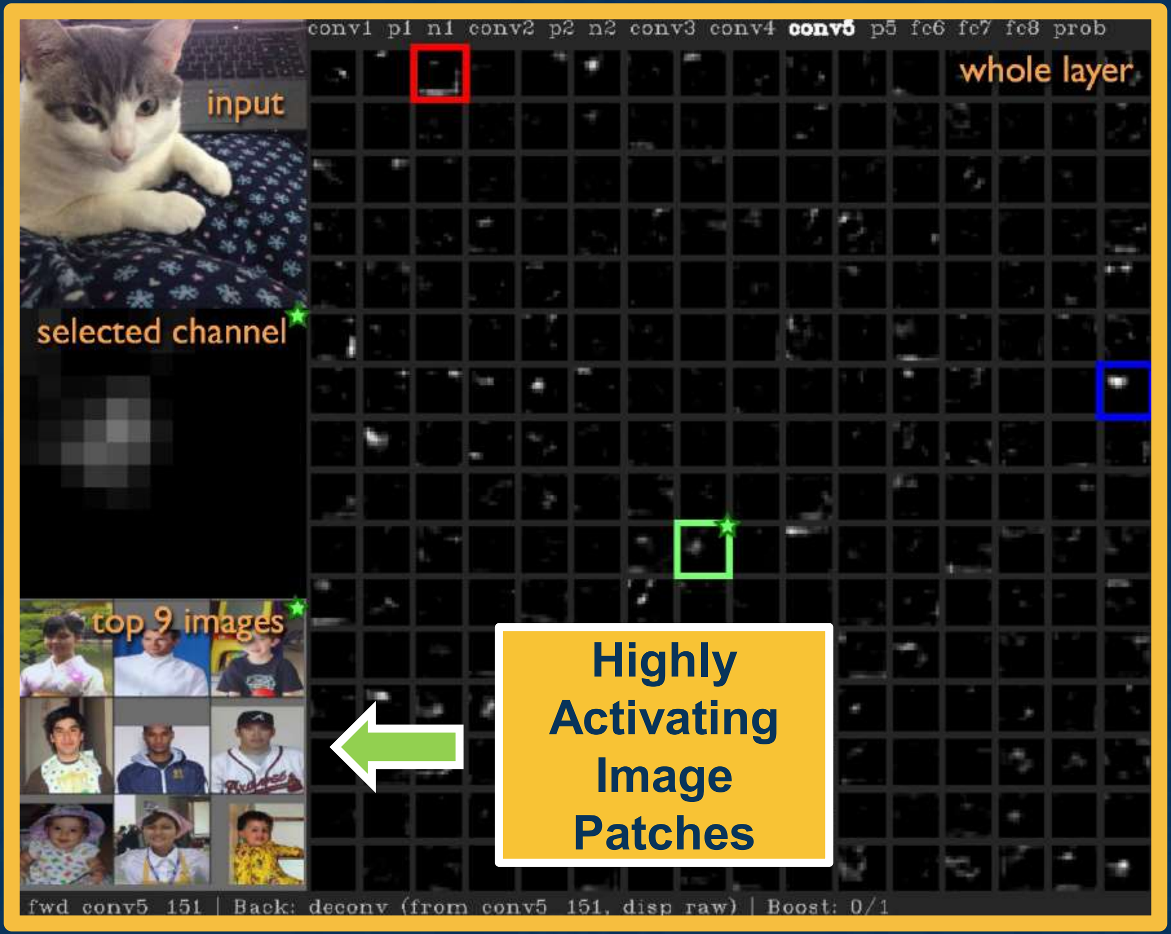
These are **larger** early in the network.



Visualizing Output Maps

Georgia Tech

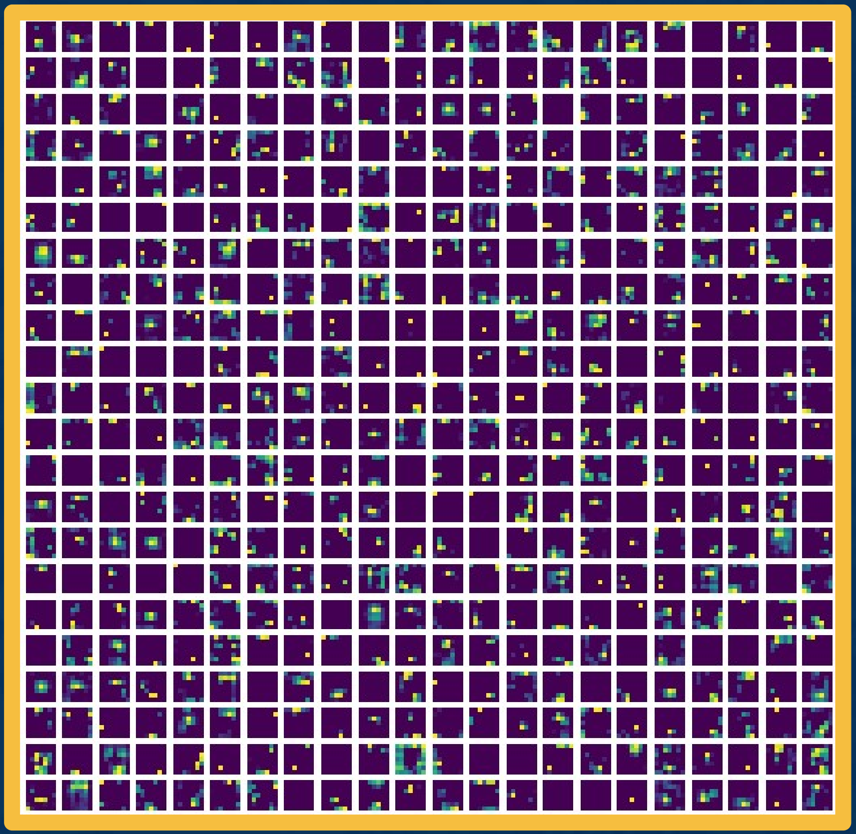
Visualizing Output Maps



From: Yosinski et al., "Understanding Neural Networks Through Deep Visualization", 2015

Georgia Tech

Activations – Small Output Sizes

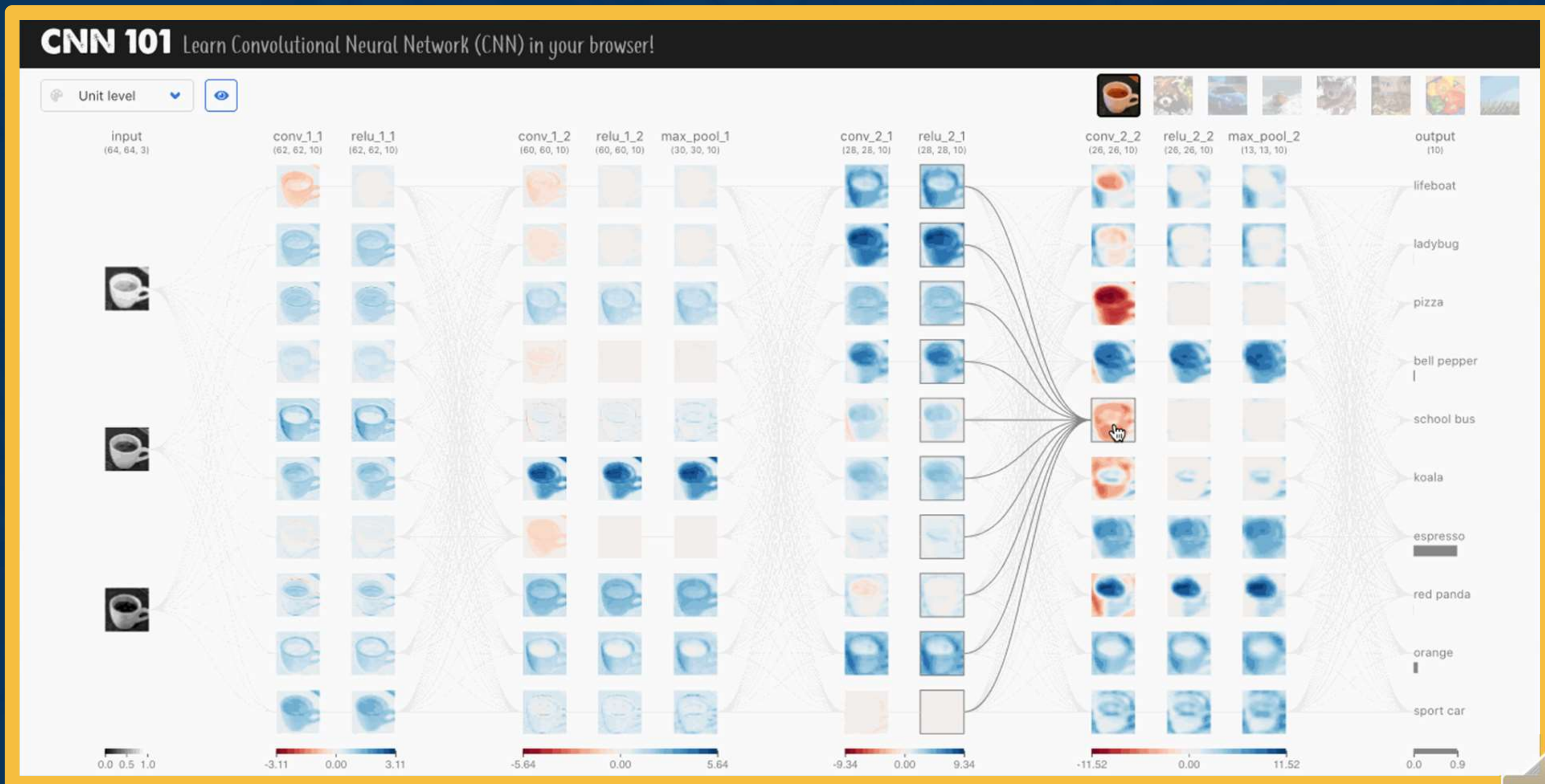


Activations of last conv layer in VGG network

Problem: Small conv outputs also hard to interpret

Georgia Tech

CNN101 and CNN Explainer



https://poloclub.github.io/cnn-explainer/

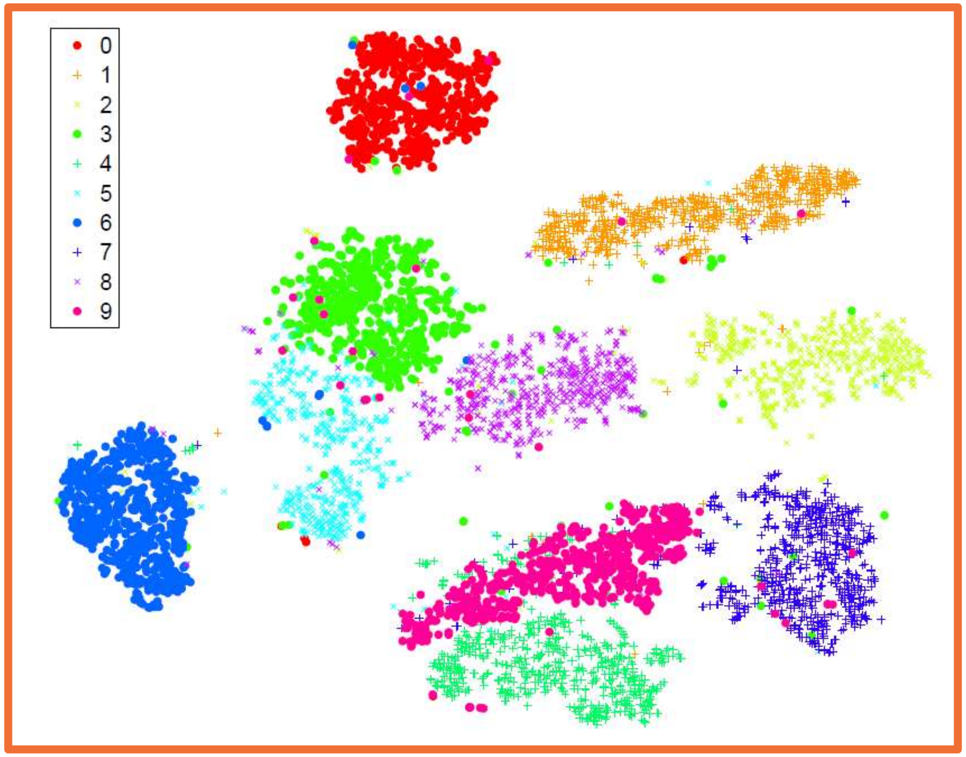
https://fredhohman.com/papers/cnn101

We can take the activations of any layer (FC, conv, etc.) and **perform dimensionality reduction**

- Often reduce to two dimensions for plotting
- E.g. using Principle Component Analysis (PCA)

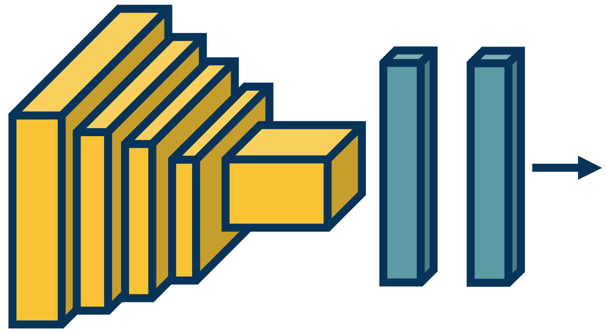
t-SNE is most common

- Performs non-linear mapping to preserve pair-wise distances



Van der Maaten & Hinton, "Visualizing Data using t-SNE", 2008.

Dimensionality Reduction: t-SNE



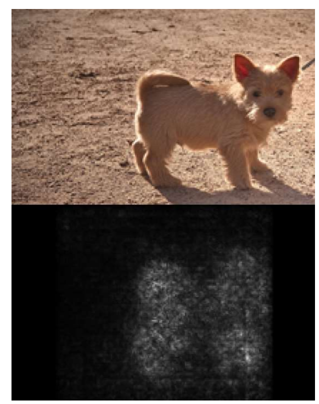
Weights

Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n

Zeiler & Fergus, 2014

Activations

Gradients



Simonyan et al, 2013

Robustness

Hendrycks & Dietterich, 2019

Visualizing Neural Networks

Summary & Caveats

While these methods provide **some** visually interpretable representations, they can be misleading or uninformative (Adebayo et al., 2018)

Assessing interpretability is difficult

- Requires **user studies** to show **usefulness**
- E.g. they allow a user to predict mistakes beforehand

Neural networks learn **distributed representation**

- (no one node represents a particular feature)
- This makes interpretation difficult

Adebayo et al., "Sanity Checks for Saliency Maps", 2018.